常州大学

实 习 报 告

题　　目　字符界面小游戏——鲲宠大乐斗

学　　院　信息数理学院

专业班级　物联网 161

学生姓名　钱宇超

学　　号　15488122

指导老师　陈哲

实习时间　2018 年 5 月 21 日　～　2018 年 5 月 25 日

目录

1 实习目的

利用已有的C语言知识，进一步学习Windows API相关知识，通过编写一个字符界面小游戏来实践C语言在Windows下的开发过程，学习使用Windows API中操作控制台的相关接口，例如`GetConsoleCursorInfo`、`SetConsoleCursorPosition`、`WriteConsole`等，同时体会开发软件时「解耦」的设计理念，熟悉各模块分离的软件架构方式，以及体验**团队分工**形式的软件开发流程。

2 实习内容

2.1 问题分析和操作逻辑设计

综合根据实训课教学代码的实现和自身对该游戏的理解，小游戏启动后需经过一个加载界面（由于只是实习目的，这里实际并不需要加载游戏内容，而是仅仅显示一个加载动画），持续若干秒后启动到游戏主菜单，这里会进入一个键盘事件循环，不断尝试读取用户的键盘输入，一旦有输入，就根据键码来执行不同的操作，如按下「下方向键」会高亮下一个菜单项、按下回车会选中菜单项。用户在主菜单按下回车或空格后，会根据菜单项的不同进入不同的次级菜单，或进入其它界面，在次级菜单中也同样有键盘事件循环，操作逻辑和主菜单相似。

为了保持界面的一致性和开发的便捷性，在不影响实践效果的情况下，决定将主要的游戏界面也设计为和主菜单类似的样式，也就是说，游戏的功能也通过菜单的形式让用户选择，设想的游戏方式类似于网络上的一款HTML5小游戏——小黑屋，如图1。

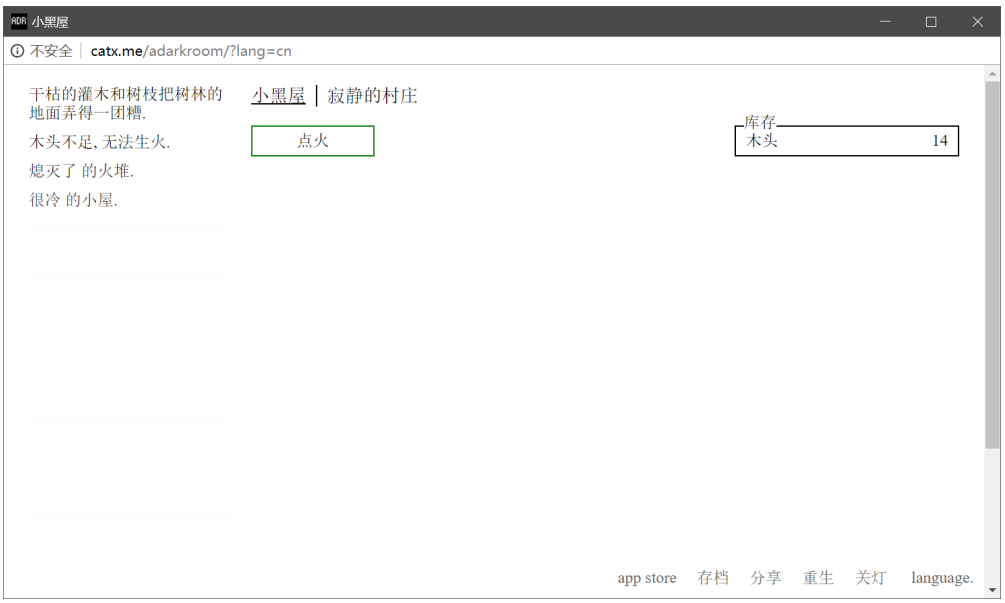

图1: 「小黑屋」游戏画面

代码实现方面，尽管教学代码中使用了C语言，这里决定使用C++来编写，得益于C++的类机制和现代C++的一些优质特性，程序会更加健壮、代码更加优雅。

2.2 界面设计

有了上小节所述的操作逻辑，开始思考界面设计。

为了保持视觉上的一致，加载界面不再使用和教学代码类似的设计，而是保持和后面的游戏界面同样的风格，在界面下方显示一个加载进度条滚动显示，如图 2。加载完成后滚动条变为一个提示文字，提示用户按空格开始游戏。

图 2: 加载界面

游戏界面方面，由于已经决定统一使用菜单进行功能的选择，那么整体只需要一种界面样式即可，界面边界处用字符画一个边框，界面上半部分显示可选的文本内容用于提示用户或显示游戏信息，下半部分显示游戏菜单，如图 3。

2.3 体验设计

游戏中宠物等级、升级机制、商店、战斗等涉及到游戏体验的内容，由小组其他成员设计，并在后续的代码实现过程中通过沟通交流进行了相应的扩展和妥协。

2.4 程序架构设计

前面的游戏逻辑和界面设计直接导致了程序架构将类似于桌面或手机应用软件，而非常见的基于物理引擎的视频游戏，经过一番考虑和妥协（由于时间相对比较紧迫，无

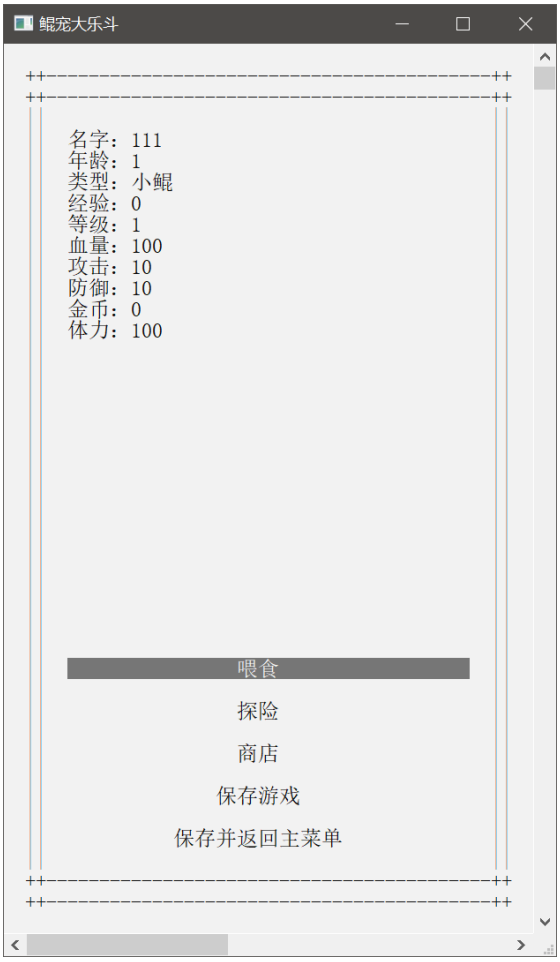

图 3: 游戏界面的统一样式

法将其设计得过于复杂），决定从手机应用开发中借鉴**视图（View）**的概念（为了方便阅读，后面将主要使用「View」这一术语），抛去其中较为复杂且在此游戏中没有必要的 View 树机制（每个 View 中又可以包括各种子 View），而是简单地将一个游戏界面实现为一个 View，每个 View 直接占据全屏，View 之间的切换将直接覆盖掉当前界面，而不会保留上一层 View 所绘制的画面，当新的 View 退出、上一层 View 重新显示时，它将会重新绘制自身所需显示的内容。

有了 View 这样一个概念之后，可以将程序划分为四个模块：`engine`、`views`、`core`、`utils`。其中，`engine` 模块实现了底层的界面绘制细节，并提供了几个基础的 `View` 类；`views` 模块中，每个游戏界面对应一个 `View` 子类，用于实现游戏的操作逻辑；`core` 模块提供了游戏的核心类，例如宠物类、战斗角色类、游戏存档和载入等；`utils` 模块则提供零碎的、和主要程序不相关的辅助工具。下面将对这些模块的具体实现逐一做详细说明。

2.5 程序实现

2.5.1 engine 模块

`engine` 模块是整个游戏程序的基石，在程序架构设计得差不多之后，就需要开始实现此模块。

首先需要定义若干个与坐标和图形相关的基础类，具体而言包括 `Point`、`Size`、`Rect`、`Color`，这些类定义在 `types.h` 头文件中，依次分别表示坐标上的点（包括 X 和 Y 坐标）、大小范围（包括宽和高）、矩形（包括两个可以确定矩形位置和大小的顶点）、颜色。代码片段 1 给出了 `Size` 类的实现，其它类基本与其相似。

然后在 `engine.h` 头文件中定义了 `Screen` 类，这个类封装了 Windows API 中对控制台的操作函数，对外提供了设置当前颜色、填充控制台一块矩形区域的颜色、从某一坐标开始打印字符串等接口，如代码片段 2。

由于 `Screen` 类只是提供了一个基本的封装，对外提供的接口仍然较为底层，直接使用它来绘制 View 仍然较为麻烦，于是在它只上又封装了一个 `Canvas` 类，该类内部保存一个 `Screen` 类的引用，然后提供更为高层的接口，如绘制边框、在矩形范围内绘制文本（自动处理换行）、使用属性绘制文本（可设置对齐方式、边距大小等），代码片段 3 展示了在矩形范围内绘制文本接口的实现。

有了上面两个绘制相关的类之后，开始实现最重要的 `View` 类，这个类虽然重要，但代码较为简单，主要就是定义了一个 View 的生命周期——何时被绘制、何时被重绘、何时进入主循环。`View` 类是一个 **抽象基类**，在实际使用时，会从它继承一个具体的类来绘制相应的游戏界面，代码片段 4 展示了 `View` 类的几个重要方法。

由于菜单是一个较为独立的组件，它有自己的特性，比如，需要在循环中接受用户的键盘输入、动态改变菜单项的高亮、在回车时执行不同的操作，实现中将它单独作为一个类来实现，即 `Menu` 类。`Menu` 类在每次循环开始的时候在给定的矩形范围内输出菜单项，并进入循环接收用户键盘输入，当收到键盘输入时，它会负责响应上下左右方向键，以上移或下移当前高亮的菜单项，而当输入是空格或回车时，它会调用创建菜单时传入的回调函数，并将当前选中的项的索引传入回调函数。`Menu` 类的具体实现不再给出，相对

的，代码片段 5 给出了在 View 子类中使用 Menu 类来输出菜单的一个例子——MenuView 类的实现。

MenuView、TextView、NoticeView 是 engine 模块提供的三个 View 子类，分别用于提供显示菜单的视图、显示文本和菜单的视图、显示通知的视图（文本和仅有的一个菜单项）。利用这些预设的子类，游戏界面的实现将会更加简单方便。

2.5.2 views 模块

该模块中全部是游戏界面的 View,例如 GameDashboardView、AboutView、BattleView 等。它们全都继承自 View 类或 engine 模块预设的 View 子类，然后覆盖父类的相应方法来实现不同的功能。以 BattleView 为例，它继承自 TextView，覆盖了 std::string text()、std::vector<std::string> menus()、void on_select(Menu &menu, const int index) 三个方法来分别定制界面上半部分显示的文本、界面下半部分显示的菜单项、选中菜单项之后的回调。得益于 View 类和 TextView 类的抽象，在 BattleView 中完全不需要关注菜单如何显示、键盘如何选择等细节，而只需要专注于游戏控制逻辑的开发。

2.5.3 core 模块

该模块提供游戏核心逻辑所需的类，由于这是一个宠物养成类的游戏，必须有一个类来表示宠物，其中包括宠物名字、年龄、等级、各种属性等，这些在 Pet 类中定义。

除此之外该模块还提供了游戏存档和载入功能，以及战斗时的敌人和宠物两个角色的统一类 BattleSpirit。

2.5.4 游戏的启动

总经过上面各模块的封装和抽象之后，启动游戏的代码如代码片段 6 所示，优雅而美丽。

3 实习总结

这五天的实习增进了自己对 Windows API 的熟悉和理解，对控制台字符界面程序有了新的体验，以前只是去用别人写的控制台程序，自己去实现的时候是有不一样的感受。另外，View 机制的实现也着实费了一些功夫，一方面，要实现得像手机应用那样复杂是没有时间和精力的，另一方面，如果完全不去做 View 机制，会使代码非常繁杂，不能很好的复用，从这里面的权衡之中还是有所收获的。

4 思考与展望

由于时间非常有限，虽说是五天实习，但实际开发时间只有两天左右，有很多细节考虑不是很到位，比如 Screen 类和 Canvas 类之间有很多功能上的重复，具体的接口设

计也不是很明确区分，以及，View 的跳转机制虽然可以工作，但还是有些缺陷，如果要实现更复杂的界面，有可能还需要优化代码结构。

另外，大部分精力花在了程序架构上面，游戏实际功能做得比较少，有很多想法没有时间去落实。

以后如果有机会可以再次重新思考这样的程序的写法，实现一个更完善的试图机制，更可以实现一个物理引擎，这样它就同时具备了写类似应用的程序和类似游戏的程序的能力。

```cpp
using Int = int16_t;

struct Size {
    Size() = default;
    Size(const Int width, const Int height) : width(width), height(height) {}

    Int width = 0;
    Int height = 0;

    bool operator==(const Size &other) const { return width == other.width && height == other.height; }
    bool operator<(const Size &other) const { return width * height < other.width * other.height; }
    bool operator<=(const Size &other) const { return *this < other || *this == other; }
};
```

代码片段 1: `Size` 类的实现

```
class Screen {
    // ...

    Screen &show_cursor() {
        CONSOLE_CURSOR_INFO cursor_info;
        GetConsoleCursorInfo(handle_, &cursor_info);
        cursor_info.bVisible = TRUE;
        SetConsoleCursorInfo(handle_, &cursor_info);
        return *this;
    }

    Screen &fill_color(const Color color, Rect rect = Rect(-1, -1, -1,
    ↪  -1)) {
        const auto buffer_info = get_buffer_info();
        if (rect == Rect(-1, -1, -1, -1)) {
            rect = Rect({0, 0}, get_size());
        }
        DWORD num;
        const auto attr =
        ↪  attribute_with_new_color(buffer_info.wAttributes, color);
        for (auto y = rect.top(); y <= rect.bottom(); y++) {
            FillConsoleOutputAttribute(handle_,
                                       attr,
                                       rect.width() * 2 /* 控制台里的实际
                                       ↪  宽度是字符数的两倍 */,
                                       COORD{rect.left() * 2, y},
                                       &num);
        }
        return *this;
    }

    // ...

    Screen &write(const std::string &str) {
        DWORD num;
        WriteConsoleA(handle_, str.c_str(), str.size(), &num, nullptr);
        return *this;
    }

    // ...
};
```

代码片段 2: Screen 类的实现

```cpp
class Canvas {
    // ...

    void draw_text(const std::string &text, const Rect &boundary) {
        std::vector<std::string> text_lines;
        std::istringstream iss(text);
        std::string line;
        while (std::getline(iss, line)) {
            text_lines.push_back(line);
        }

        const auto text_size_per_line = boundary.width() * 2;

        auto pos = boundary.top_left();
        for (size_t i = 0; i < text_lines.size() && pos.y <= boundary.bottom(); i++) {
            if (text_lines[i].empty()) {
                // 当前文本行是空行，直接输出一排空白
                screen_.write(pos, std::string(text_size_per_line, ' '));
                pos.y++;
            } else {
                // 输出当前文本行
                for (Int j = 0; pos.y <= boundary.bottom() && text_size_per_line * j < text_lines[i].size();
                        j++, pos.y++) {
                    auto line_in_rect = text_lines[i].substr(text_size_per_line * j, text_size_per_line);
                    line_in_rect.resize(text_size_per_line, ' '); // 填充当前矩形行剩下的空间
                    screen_.write(pos, line_in_rect);
                }
            }
        }
    }

    // ...
}
```

代码片段 3: Canvas 类的在矩形范围内绘制文本接口

```cpp
class View {
public:
    // ...

    // 在指定控制台上显示当前 View
    virtual void display_on(Screen &screen) {
        canvas_ = std::make_shared<Canvas>(screen);
        redraw();
        loop();
    }

protected:
    // ...

    // 绘制 View 的静态内容，留给子类实现
    virtual void draw() {
        canvas_->fill_and_set_background_color(Colors::WHITE);
        canvas_->set_foreground_color(Colors::BLACK);
        canvas_->draw_border({1, 1, 1, 1}, 2);
    }

    // 主循环，子类应该自行在重写方法中实现相应的循环
    // 当此函数返回后，其所属的 View 就不再有效了，如果是通过 jump 显示的，将会回到上一级页面的控制
    virtual void loop() {}

    // 清屏之后再绘制
    void redraw() {
        canvas_->clear();
        draw();
    }

    void jump(View &new_view) {
        // 显示新页面，进入它的主循环
        new_view.display_on(canvas_->get_screen());

        // 从下一个页面返回，重新绘制当前页面，
        // 如果当前页是菜单视图，那实际上 jump 是在 on_select 里面调用的，
        // 这里重绘页面之后，on_select 返回，Menu 类会自己重新输出菜单项
        redraw();
    }
};
```

代码片段 4: `View` 类的几个重要方法

```
class MenuView : public View {
protected:
    void draw() override { View::draw(); }

    void loop() override {
        Menu menu(*canvas_, menus(), [this](Menu &menu, const int index)
        ↪  { on_select(menu, index); });
        menu.show(get_inner_boundary().expanded(-2));
    }

    virtual std::vector<std::string> menus() = 0;
    virtual void on_select(Menu &menu, int index) = 0;
};
```

代码片段 5: MenuView 类的实现

```
void start() {
    Screen screen(GetStdHandle(STD_OUTPUT_HANDLE));
    screen.set_size({25, 42});
    SetConsoleTitleA(" 鲲宠大乐斗");

    views::LoadingView view;
    view.display_on(screen);
}
```

代码片段 6: 启动游戏的代码